L Number	Hits	Search Text	DB	Time stamp
-	7	087853.ap.	USPAT; US-PGPUB	2004/08/02 17:10
-	368	application$1 with socket$1 with port$1	USPAT; US-PGPUB	2004/08/02 17:30
-	281	(application$1 with socket$1 with port$1) and @ad<20020304	USPAT; US-PGPUB	2004/08/03 15:00
-	169	((application$1 with socket$1 with port$1) and @ad<20020304) and ((IP or Internet) adj3 address$3)	USPAT; US-PGPUB	2004/08/02 17:32
-	161	(((application$1 with socket$1 with port$1) and @ad<20020304) and ((IP or Internet) adj3 address$3)) and (single or unique)	USPAT; US-PGPUB	2004/08/02 17:33
-	282	(application$1 with socket$1 with port$1) and @ad<20020304	USPAT; US-PGPUB	2004/08/03 15:02
-	0	((application$1 with socket$1 with port$1) and @ad<20020304) and (enumerat$3 adj2 port$1)	USPAT; US-PGPUB	2004/08/03 15:01
-	71	enumerat$3 adj2 port$1	USPAT; US-PGPUB	2004/08/03 15:02
-	49	(enumerat$3 adj2 port$1) and @ad<20020304	USPAT; US-PGPUB	2004/08/03 15:03
-	3	((enumerat$3 adj2 port$1) and @ad<20020304) and (ip adj2 address$3)	USPAT; US-PGPUB	2004/08/03 15:03

C:\APPS\EAST\Workspaces\10-087853 Route to selected system based on PORT info..wsp

DOCUMENT-IDENTIFIER: **US 20020143965 A1**

TITLE: Server application initiated affinity within networks performing workload balancing

---------- KWIC ---------

Pre-Grant Publication Document Identifier - DID (1):
US 20020143965 A1

Application Filing Date - APD (1):
20010403

Summary of Invention Paragraph - BSTX (7):
[0006] The Internet Protocol ("IP") is designed as a connectionless protocol. Therefore, IP workload balancing solutions treat every Transmission Control Protocol ("TCP") connection request to a particular application, identified by a particular destination **IP address** and port number combination, as independent of all other such TCP connection requests. Examples of such IP workload balancing systems include Sysplex Distributor from the International Business Machines Corporation ("IBM"), which is included in IBM's OS/390.RTM. TCP/IP implementation, and the Multi-Node Load Balancer ("MNLB") from Cisco Systems, Inc. ("OS/390" is a registered trademark of IBM.) Workload balancing solutions such as these use relative server capacity (and, in the case of Sysplex Distributor, also network policy information and quality of service considerations) to dynamically select a server to handle each incoming connection request. However, some applications require a relationship between a particular client and a particular server to persist beyond the lifetime of a **single** interaction (i.e. beyond the connection request and its associated response message).

Summary of Invention Paragraph - BSTX (10):
[0009] The need to provide these persistent relationships is often referred to as "server affinity" or "the sticky routing problem". One technique that has been used in the prior art to address this problem for web applications is use of "cookies". A "cookie" is a data object transported in variable-length fields within HTTP request and response headers. A cookie stores certain data that the server application wants to remember about a particular client. This could include client identification, parameters and state information used in an on-going transaction, user preferences, or almost anything else an application writer can think of to include. Cookies are normally stored on the client device, either for the duration of a transaction (e.g. throughout a customer's electronic shopping interactions with an on-line merchant via a **single** browser instance) or permanently. A web application may provide identifying information in the cookies it transmits to clients in response messages, where the client then returns that information in subsequent request

messages. In this manner, the client and server application make use of connection-oriented information in spite of the connection-less model on which HTTP was designed.

Summary of Invention Paragraph - BSTX (12):
[0011] Other types of applications may have solutions to the sticky routing problem that depend on client and server application cooperation using techniques such as **unique** application-specific protocols to preserve and transfer relationship state information between consecutive connection lifetimes. For example, the Lotus Notes.RTM. software product from Lotus Development Corporation requires the client application to participate, along with the server application, in the process of locating the proper instance of a server application on which a particular client user's e-mail messages are stored. ("Lotus Notes" is a registered trademark of Lotus Development Corporation.) In another cooperative technique, the server application may transmit a special return address to the client, which the client then uses for a subsequent message.

Summary of Invention Paragraph - BSTX (14):
[0013] The sticky routing problem is further complicated by the fact that multiple TCP connections are sometimes established in parallel from a **single** client, so that related requests can be made and processed in parallel (for example, to more quickly deliver a web document composed of multiple elements). A typical browser loads up to four objects concurrently on four simultaneous TCP connections. In applications where state information is required or desirable when processing parallel requests, the workload balancing implementation cannot be allowed to independently select a server to process each connection request.

Summary of Invention Paragraph - BSTX (15):
[0014] One prior art solution to the sticky routing problem in networking environments which perform workload balancing is to establish an affinity between a client and a server by configuring the workload balancing implementation to perform special handling for incoming connection requests from a predetermined client **IP address** (or perhaps a group of client **IP addresses** which is specified using a subnet address). This configuring of the workload balancer is typically a manual process and one which requires a great deal of administrative work. Because it is directed specifically to a known client **IP address** or subnet, this approach does not scale well for a general solution nor does it adapt well to dynamically-determined client **IP addresses** which cannot be predicted accurately in advance. Furthermore, this configuration approach is static, requiring reconfiguration of the workload balancer to alter the special defined handling. This static specification of particular client addresses for which special handling is to be provided may result in significant workload imbalances over time, and thus this is not an optimal solution.

Summary of Invention Paragraph - BSTX (16):
[0015] In another approach, different target server names (which are

resolved to server **<u>IP addresses</u>**) may be statically assigned to client populations. This approach is used by many nation-wide Internet Service Providers ("ISPs"), and requires configuration of clients rather than servers.

Summary of Invention Paragraph - BSTX (17):
[0016] Another prior art approach to the sticky routing problem in networking environments which perform workload balancing is to use "timed" affinities. Once a server has been selected for a request from a particular client **<u>IP address</u>** (or perhaps from a particular subnet), all subsequent incoming requests that arrive within a predetermined fixed period of time (which may be configurable) are automatically sent to that same server. However, the dynamic nature of network traffic makes it very difficult to accurately predict an optimal affinity duration, and use of timed affinities may therefore result in serious inefficiencies and imbalances in the workload. If the affinity duration is too short, then the relationship may be ended prematurely, If the duration is too long, then the purpose of workload balancing is defeated. In addition, significant resources may be wasted when the affinity persists after it is no longer needed.

Summary of Invention Paragraph - BSTX (30):
[0028] To achieve the foregoing objects, and in accordance with the purpose of the invention as broadly described herein, the present invention provides methods, systems, and computer program products for handling on-going relationships requiring multiple exchanges of related requests over a communications network in the presence of workload balancing. In a first aspect of one embodiment, this technique comprises: providing server affinities for related connection request messages, comprising: signaling, by an executing server application, that an affinity with a selected source is to be started; and bypassing normal workload balancing operations, responsive to the signaling, for subsequent connection request messages from the selected source while the affinity persists. The selected source may be a selected client, in which case the selected client may be identified by its **<u>IP address</u>** or perhaps by its **<u>IP address</u>** and port number. Or, the selected source may be a selected client subnetwork.

Detail Description Paragraph - DETX (11):
[0049] Preferably, each of the communication protocol stacks 22, 26, 30, 34, 38 has associated therewith a list of addresses (such as **<u>IP addresses</u>**) for which that stack is responsible. Also, each data processing system 20, 24, 28, 32 or MVS image preferably has associated therewith a **<u>unique</u>** identifier within the sysplex 10. At initialization of the communication protocol stacks 22, 26, 30, 34, 38, the stacks are preferably configured with the addresses for which that stack will be responsible, and are provided with the identifier of the MVS image of the data processing system.

Detail Description Paragraph - DETX (12):
[0050] Note that while destination addresses within the sysplex are referred to herein as **<u>"IP" addresses,</u>** these addresses are preferably a virtual **<u>IP address</u>** of some sort, such as a Dynamic Virtual **<u>IP Address</u>** ("DVIPA") of the

type described in U.S. Pat. ______ (Ser. No. 09/640,409), which is assigned to IBM and is entitled "Methods, Systems and Computer Program Products for Cluster Workload Distribution", or a loopback equivalent to a DVIPA, whereby the address appears to be active on more than one stack although the network knows of only one place to send IP packets destined for that **IP address**. As taught in the DVIPA patent, an **IP address** is not statically defined in a configuration profile with the normal combination of DEVICE, LINK, and HOME statements, but is instead created as needed (e.g. when needed by Sysplex Distributor),

Detail Description Paragraph - DETX (15):
[0053] In the first preferred embodiment, the server application explicitly informs the workload balancing function when a relationship with a particular client starts (as will be described in more detail below, with reference to FIG. 4). Preferably, the client is identified on this "start affinity" message by its **IP address**. One or more port numbers may also be identified, if desired. When port numbers are specified, the workload balancing function is bypassed only for connection requests originating from those particular ports; if port numbers are omitted, then the workload balancing function is preferably bypassed for connection requests originating from all ports at the specified client source **IP address**. In this preferred embodiment, the start affinity notification (as well as an optional end affinity message) is preferably sent from the application to its hosting stack, which forwards the message to the workload balancing function. (Hereinafter, a communication protocol stack on which one or more server applications execute is referred to as a "target stack", a "hosting stack", or a "target/hosting stack". A particular stack may be considered a "target" from the point of view of the workload balancer, and a "host" from the point of view of a server application executing on that stack, or both a target and a host when both the workload balancer and a server application are being discussed.)

Detail Description Paragraph - DETX (18):
[0056] When used as a start affinity request, message format 200 uses fields 202, 204, 206, 208, 210, 212, and 214; fields 216 and 218 are unused. The local **IP address** field 202 preferably specifies the **IP address** for which an affinity is being established. The local port number field 204 specifies the port number of the **IP address** for which this affinity is to be established. If port number field 204 is zero, then all connection requests arriving at the listening socket (see field 214) are covered by this affinity. If the port number field 204 contains a non-zero value, then the affinity applies only to connection requests arriving for that particular port.

Detail Description Paragraph - DETX (19):
[0057] The partner **IP address** field 206 specifies the source **IP address** of the client to be covered by this affinity. In an optional enhancement, a range of client addresses may be specified for affinity processing. (This enhancement is referred to herein as "affinity group" processing.) In this case, the partner **IP address** field 206 specifies a subnet address, and a subnet mask or prefix field 208 is preferably used to indicate how many **IP addresses** are to be covered. (If the high-order bit is "1", this indicates a subnet mask

of field 208 indicates how many "1" bits are to be used for the subnet mask.) The partner port number field 210 may specie a particular port number to be used for the affinity, or alternatively may be zero to indicate that the affinity applies to any connection request from the partner **IP address**. (In an alternative embodiment, multiple port numbers may be supported, for example by specifying a comma-separated list of values in field 210.)

Detail Description Paragraph - DETX (21):
[0059] The following verification is preferably performed on the values of the start affinity request message: (1) The local **IP address** value 202 must be a valid **IP address** for which the hosting stack is a valid target for at least one port. (2) The local port value 204, when non-zero, must match an established listening socket. (3) The partner **IP address** 206 must be non-zero. (4) The partner/mask prefix 208 must be non-zero. (5) If the duration 208 exceeds the default maximum for the hosting stack, then the specified value in field 208 will be ignored. (6) If the socket is bound to a specific **IP address**, it must be the same as the local **IP address** in field 202.

Detail Description Paragraph - DETX (22):
[0060] When used as a start affinity response, message format 200 uses all fields shown in FIG. 2A. Most fields are simply copied from the corresponding request message when generating the response message; however, several of the fields are used differently, as will now be described. First, if the local port number 204 was zero on the request message, it will be filled in with an actual port number on the response, as determined by the listening socket handle. Second, the return code field 216 is set, and may indicate a successful start affinity or an unsuccessful start, or perhaps a successful start with a warning message. Finally, the additional information field 218 is set, and preferably conveys additional information about the return code value 216. Preferably, **unique** field value encodings are defined for one or more of the following cases: affinity successfully created; affinity successfully renewed; warning that affinity was not established as requested, and clock was not restarted, because the requested affinity falls within an overlapping affinity for a smaller prefix or larger subnet for which an affinity already exists; unsuccessful because the hosting stack is not a target stack for the specified local **IP address**; unsuccessful because the requested port does not match the listening socket; unsuccessful because the socket is not a valid listening socket; and unsuccessful because an affinity with the partner **IP address** was already established by another requester.

Detail Description Paragraph - DETX (23):
[0061] Referring now to FIG. 2B, when used as a start affinity request from the hosting stack to the workload balancer, message format 220 uses fields 222, 224, 226, 228, 230, and 232; fields 234 and 236 are unused. Fields 222, 224, 226, 228, and 230 are preferably copied by the hosting stack from the corresponding fields 202, 204, 206, 208, and 210 which were received from the application on its start affinity request message. The local **port** number 224, however, may either have been supplied by the **application** or copied from the listening **socket** information 214. Stack identity 232 identifies the hosting stack to which the subsequent connections covered by the affinity should be

sent. The specified value could be a **unique** names within the sysplex (such as an operating system name and a job name within that operating system), or a **unique** address such as an **IP address**; what is required is that the provided identity information suffices to uniquely identify the stack that will handle the incoming connection requests, even if the are multiple stacks per operating system image (such as stacks 34 and 38 in FIG. 1).

Detail Description Paragraph - DETX (24):
[0062] When used as a start affinity response, message format 220 uses all fields shown in FIG. 2B. Preferably, fields 222 through 232 are simply copied from the corresponding request message when generating the response message. The return code field 234 is set in the response, and may indicate a successful start affinity or an unsuccessful start, or a successful start with a warning message. The additional information field 236 is also set, and preferably conveys additional information about the return code value 234. Preferably, **unique** field value encodings are defined for one or more of the following cases: affinity successfully created; affinity successfully renewed; and unsuccessful because an affinity with the partner **IP address** was already established by another requester.

Detail Description Paragraph - DETX (25):
[0063] Preferably, existing affinities that are known to the workload balancing function are stored in a table or other similar structure, such as that illustrated in FIG. 3A. For purposes of illustration but not of limitation, the affinity table may be organized according to the destination server application. As shown in FIG. 3A, the server application type 305 of affinity table 300 preferably comprises (1) the **IP address** 310 of the server application (which corresponds to the destination **IP address** of incoming client connection requests) and (2) the port number 315 of that server application (which corresponds to the destination port number of the incoming client connection requests). These values are taken from fields 222 and 224 of start affinity request messages 220 (FIG. 2B). Preferably, if a server application uses multiple ports, then a separate entry is created in affinity table 300 for each such port. (Alternatively, a list of port numbers may be supported in field 315.)

Detail Description Paragraph - DETX (27):
[0065] Each server application identified by an entry in fields 310, 315 may have an arbitrary number of client affinity entries 325. Each such client affinity entry 325 preferably comprises (1) the client's **IP address** 330 (which corresponds to the source **IP address** of incoming client connection requests), (2) a subnet mask or prefix value 335, which is used for comparing incoming client **IP addresses** to source **IP address** 330 using known techniques, and (3) optionally, the port number 340 of the client application (which corresponds to the source port number of the incoming client connection requests). These values are taken from fields 226, 228, and 230 of start affinity request messages 220 (FIG. 2B). If the client port number is omitted from a particular start affinity message or is set to zero, indicating that an affinity is defined for all ports from a particular client (as discussed above with reference to FIG. 2A), then a port number of zero is preferably used in field

340 to indicate that all ports are to be considered as matching. Alternatively, the port number field 340 may be left blank, or a special keyword such as "ALL" or perhaps a wildcard symbol such as "*" may be provided as the field value. If multiple client port numbers are specified on the start affinity message, then values for the port number field 340 are preferably stored using a comma-separated list (or perhaps an array or a pointer thereto). In an alternative approach, a separate record might be created in the affinity table for each different client port number.

Detail Description Paragraph - DETX (28):
[0066] The table 350 shown in FIG. 3B illustrates a structure that may be used by hosting stacks to manage their existing affinities. As with the table used by the workload balancer and illustrated in FIG. 3A, entries in the affinity table 350 of FIG. 3B may be organized according to the destination server application. Thus, the server application type 355 of affinity table 350 preferably comprises (1) the **IP address** 360 of the server application and (2) the port number 365 of that server application. These values are taken from fields 202 and 204 of start affinity request messages 200 (FIG. 2A). (Even though the **IP address and port** number of the server **application** are contained in the **socket** control block at the hosting stack, they are preferably stored in the affinity entries as well for efficiency in matching against incoming connection requests.) Preferably, if a server **application** uses multiple **ports,** then a separate entry is created in affinity table 350 for each such **port**.

Detail Description Paragraph - DETX (36):
[0074] When used as an end affinity response from a hosting stack to an application, message format 240 uses all fields shown in FIG. 2C. Fields 242 through 254 are preferably copied from the corresponding request message when generating the response message. The return code field 256 may indicate a successful end affinity or an unsuccessful end. The additional information field 258 is set, and preferably conveys additional information about the return code value 256. Preferably, **unique** field value encodings are defined for one or more of the following cases: affinity successfully ended; unsuccessful, affinity not ended because the requested affinity falls within an overlapping affinity for a smaller prefix or larger subnet for which an affinity already exists; and unsuccessful because a matching affinity was not found.

Detail Description Paragraph - DETX (37):
[0075] When used as an end affinity indication from a hosting stack to an application, message format 240 uses all fields described for the end affinity response, except that field 256 is not meaningful, and field 258 now contains additional information about the reason for the unsolicited indication message. The additional information field 258 preferably uses **unique** field value encodings for one or more of the following cases to explain why an affinity was ended: timer expiration; the local **IP address** is no longer valid; hosting stack is no longer a target stack for the local **IP address**; and the listening socket was closed.

Detail Description Paragraph - DETX (39):
[0077] When used as an end affinity response from the workload balancer to the hosting stack, message format 260 uses all fields shown in FIG. 2D. Preferably, fields 262 through 272 are simply copied from the corresponding request message when generating the response message. The return code field 274 is set in the response, and may indicate a successful end affinity or an unsuccessful end. The additional information field 276 is also set, and preferably conveys additional information about the return code value 274. Preferably, **unique** field value encodings are defined for one or more of the following cases: affinity successfully ended; unsuccessful end because the specified affinity falls within an affinity for a smaller prefix or larger subnet for which an affinity already exists, and unsuccessful because matching affinity could not be found.

Detail Description Paragraph - DETX (40):
[0078] When used as an end affinity indication from the workload balancer to the hosting stack, message format 260 uses all fields described for the end affinity response, except that field 274 is not meaningful, and field 276 now contains additional information about the reason for the unsolicited indication message. The additional information field 276 preferably uses **unique** field value encodings for one or more of the following cases to inform the hosting stack why the affinity is being ended: the local **IP address** is no longer valid; and the hosting stack is no longer a target stack for the local **IP address**.

Detail Description Paragraph - DETX (52):
[0089] The start affinity message may be sent from the server application to its local hosting stack over a "control socket". As used herein, a control socket is a bi-directional communications socket established between a server application and its hosting stack, Preferably, a server application establishes this control socket when it initializes, along with the normal server listening socket that is used for receiving client requests. However, the control socket provides a control conduit to the server application's hosting TCP/IP stack, rather than a communication vehicle to other applications. Preferably, the destination **IP address and port** number of the server **application** are provided as parameters when establishing the control **socket**. Once the control socket is established, the start affinity message (see Block 420), as well as any subsequent end affinity message, is preferably transmitted using that control socket.

Detail Description Paragraph - DETX (54):
[0091] Block 525 then checks to see if it is necessary to notify the workload balancer that this affinity has been started. If the affinity is new (as contrasted to an existing affinity for which a subsequent affinity request has arrived, and which is therefore being renewed by restarting the duration timer), then this test has a positive result and Block 540 adds this target stack's identity information (e.g. its job name and operating system name, or a **unique IP address** associated with the target stack) to a version of the start affinity message that is then forwarded (in Block 550) to the workload balancer. On the other hand, if this affinity is one which is being renewed,

and if all timer expiration processing is being handled by the hosting stack, then it is not necessary to forward a (renewing) start affinity message to the workload balancer as no new information would be communicated. In this case, the test in Block 525 has a negative result, and control preferably exits the processing of FIG. 5.

Detail Description Paragraph - DETX (62):
[0099] The logic in FIG. 7 illustrates affinity processing that may be performed when a workload balancer receives incoming client connection requests. A client request is received (Block 705) from a client application (such as client 46 of FIG. 1). The target server application is then determined (Block 710) by examining the destination **IP address** and port number. This information is compared to the workload balancer's stored affinity information (Block 715) to determine if affinities for this application have been defined. With reference to FIG. 3A, this comprises determining whether affinity table 300 has entries 310, 315 matching the destination information from the incoming connection request. If so, then the source **IP address** and port number are compared to the stored affinity information for that application. If an entry for this source **IP address** exists in field 330 of the client affinity information 325 (and matches according to the mask or prefix value stored in field 335), for the target application, and if the source port number of the incoming request either matches a port number specified in field 340 or the entry in field 340 indicates that all port numbers are to be considered as matching, then this is a client request for which a server affinity has been defined. In this case, the test in Block 720 has a positive result, and in Block 730 the target server is selected using the receiving/owning stack field 320; otherwise, when Block 715 fails to find a matching entry in the affinity table, then Block 720 has a negative result and the target server is selected (Block 725) as in the prior art (e.g. using the normal workload balancing process).

Detail Description Paragraph - DETX (79):
[0115] FIG. 11 depicts logic that may be used in the selected target/hosting stack for handling incoming client requests and determining whether port balancing should be performed. At Block 1100, an incoming client request is received. Block 1105 then locates the client **IP address** and port number, and the destination **IP address** and port number, from that request and checks to see if automatic affinity processing is activated for the target application. If not, then control transfers to Block 1120 which selects an instance as in the prior art. Otherwise, Block 1110 checks the active connections for the target application to determine whether this client already has at least one active connection to that same application. If so, then Block 1125 selects the target application instance to be the same one already in use; otherwise, Block 1120 selects an instance as in the prior art (e.g. using port balancing). In either case, Block 1130 the routes the incoming request to the selected instance, and the processing of FIG. 11 is then complete for this incoming message.

Claims Text - CLTX (4):
3. The method according to claim 2, wherein the selected client is identified by its **Internet Protocol ("IP") address**.

Claims Text - CLTX (5):
4. The method according to claim 2, wherein the selected client is identified by its **Internet Protocol ("IP") address** and port number.

DOCUMENT-IDENTIFIER: **US 20020143953 A1**

TITLE: Automatic affinity within networks performing workload balancing

---------- KWIC ---------

Abstract Paragraph - ABTX (1):
Methods, systems, and computer program products for automatically establishing an affinity for messages destined for a particular server application in a computing network, where that network performs workload balancing. A server application may specie (for example, using configuration values) that concurrent connection request messages from clients are to be routed to the same application instance, thereby bypassing normal workload balancing (as well as port balancing) that would otherwise occur among multiple application instances. This is advantageous for applications in which multiple concurrent requests from a particular client pertain to the same client operation (such as requests to deliver multiple elements of a **single** Web page). Access to server application code is not required.

Pre-Grant Publication Document Identifier - DID (1):
US 20020143953 A1

Application Filing Date - APD (1):
20010403

Summary of Invention Paragraph - BSTX (5):
[0005] The Internet Protocol ("IP") is designed as a connectionless protocol. Therefore, IP workload balancing solutions treat every Transmission Control Protocol ("TCP") connection request to a particular application, identified by a particular destination **IP address** and port number combination, as independent of all other such TCP connection requests. Examples of such IP workload balancing systems include Sysplex Distributor from the International Business Machines Corporation ("IBM"), which is included in IBM's OS/390.RTM. TCP/IP implementation, and the Multi-Node Load Balancer ("MNLB") from Cisco Systems, Inc. ("OS/390" is a registered trademark of IBM.) Workload balancing solutions such as these use relative server capacity (and, in the case of Sysplex Distributor, also network policy information and quality of service considerations) to dynamically select a server to handle each incoming connection request. However, some applications require a relationship between a particular client and a particular server to persist beyond the lifetime of a **single** interaction (i.e. beyond the connection request and its associated response message).

Summary of Invention Paragraph - BSTX (8):

[0008] The need to provide these persistent relationships is often referred to as "server affinity" or "the sticky routing problem". One technique that has been used in the prior art to address this problem for web applications is use of "cookies". A "cookie" is a data object transported in variable-length fields within HTTP request and response headers. A cookie stores certain data that the server application wants to remember about a particular client. This could include client identification, parameters and state information used in an on-going transaction, user preferences, or almost anything else an application writer can think of to include. Cookies are normally stored on the client device, either for the duration of a transaction (e.g. throughout a customer's electronic shopping interactions with an on-line merchant via a **single** browser instance) or permanently. A web application may provide identifying information in the cookies it transmits to clients in response messages, where the client then returns that information in subsequent request messages. In this manner, the client and server application make use of connection-oriented information in spite of the connection-less model on which HTTP was designed.

Summary of Invention Paragraph - BSTX (10):
[0010] Other types of applications may have solutions to the sticky routing problem that depend on client and server application cooperation using techniques such as **unique** application-specific protocols to preserve and transfer relationship state information between consecutive connection lifetimes. For example, the Lotus Notes.RTM. software product from Lotus Development Corporation requires the client application to participate, along with the server application, in the process of locating the proper instance of a server application on which a particular client user's e-mail messages are stored. ("Lotus Notes" is a registered trademark of Lotus Development Corporation.) In another cooperative technique, the server application may transmit a special return address to the client, which the client then uses for a subsequent message.

Summary of Invention Paragraph - BSTX (12):
[0012] The sticky routing problem is further complicated by the fact that multiple TCP connections are sometimes established in parallel from a **single** client, so that related requests can be made and processed in parallel (for example, to more quickly deliver a web document composed of multiple elements). A typical browser loads up to four objects concurrently on four simultaneous TCP connections. In applications where state information is required or desirable when processing parallel requests, the workload balancing implementation cannot be allowed to independently select a server to process each connection request.

Summary of Invention Paragraph - BSTX (13):
[0013] One prior art solution to the sticky routing problem in networking environments which perform workload balancing is to establish an affinity between a client and a server by configuring the workload balancing implementation to perform special handling for incoming connection requests from a predetermined client **IP address** (or perhaps a group of client **IP addresses** which is specified using a subnet address). This configuring of the

workload balancer is typically a manual process and one which requires a great deal of administrative work. Because it is directed specifically to a known client **IP address** or subnet, this approach does not scale well for a general solution nor does it adapt well to dynamically-determined client **IP addresses** which cannot be predicted accurately in advance. Furthermore, this configuration approach is static, requiring reconfiguration of the workload balancer to alter the special defined handling. This static specification of particular client addresses for which special handling is to be provided may result in significant workload imbalances over time, and thus this is not an optimal solution.

Summary of Invention Paragraph - BSTX (14):
[0014] In another approach, different target server names (which are resolved to server **IP addresses**) may be statically assigned to client populations. This approach is used by many nationwide Internet Service Providers ("ISPs"), and requires configuration of clients rather than servers.

Summary of Invention Paragraph - BSTX (15):
[0015] Another prior art approach to the sticky routing problem in networking environments which perform workload balancing is to use "timed" affinities. Once a server has been selected for a request from a particular client **IP address** (or perhaps from a particular subnet), all subsequent incoming requests that arrive within a predetermined fixed period of time (which may be configurable) are automatically sent to that same server. However, the dynamic nature of network traffic makes it very difficult to accurately predict an optimal affinity duration, and use of timed affinities may therefore result in serious inefficiencies and imbalances in the workload. If the affinity duration is too short, then the relationship may be ended prematurely. If the duration is too long, then the purpose of workload balancing is defeated. In addition, significant resources may be wasted when the affinity persists after it is no longer needed.

Summary of Invention Paragraph - BSTX (26):
[0025] To achieve the foregoing objects, and in accordance with the purpose of the invention as broadly described herein, the present invention provides methods, systems, and computer program products for handling on-going relationships requiring multiple exchanges of related requests over a communications network in the presence of workload balancing. In a first aspect of one embodiment, this technique comprises automatically providing server affinities for related concurrent connection requests, comprising: selectively activating an affinity for a particular server application; routing a first connection request to the particular server application from a selected source; and bypassing normal workload balancing operations, responsive to the selective activation, for subsequent concurrent connection requests for the particular server application from the selected source while at least one such concurrent connection request remains active. The selected source may be a selected client, in which case the selected client may be identified by its **IP address** or perhaps by its **IP address** and port number.

Detail Description Paragraph - DETX (11):
[0043] Preferably, each of the communication protocol stacks 22, 26, 30, 34, 38 has associated therewith a list of addresses (such as **IP addresses**) for which that stack is responsible. Also, each data processing system 20, 24, 28, 32 or MVS image preferably has associated therewith a **unique** identifier within the sysplex 10. At initialization of the communication protocol stacks 22, 26, 30, 34, 38, the stacks are preferably configured with the addresses for which that stack will be responsible, and are provided with the identifier of the MVS image of the data processing system.

Detail Description Paragraph - DETX (12):
[0044] Note that while destination addresses within the sysplex are referred to herein as **"IP" addresses,** these addresses are preferably a virtual **IP address** of some sort, such as a Dynamic Virtual **IP Address** ("DVIPA") of the type described in U.S. Pat. No. ______ (Ser. No. 09/640,409), which is assigned to IBM and is entitled "Methods, Systems and Computer Program Products for Cluster Workload Distribution", or a loopback equivalent to a DVIPA, whereby the address appears to be active on more than one stack although the network knows of only one place to send IP packets destined for that **IP address**. As taught in the DVIPA patent, an **IP address** is not statically defined in a configuration profile with the normal combination of DEVICE, LINK, and HOME statements, but is instead created as needed (e.g. when needed by Sysplex Distributor).

Detail Description Paragraph - DETX (15):
[0047] In the first preferred embodiment, the server application explicitly informs the workload balancing function when a relationship with a particular client starts (as will be described in more detail below, with reference to FIG. 4). Preferably, the client is identified on this "start affinity" message by its **IP address**. One or more port numbers may also be identified, if desired. When port numbers are specified, the workload balancing function is bypassed only for connection requests originating from those particular ports; if port numbers are omitted, then the workload balancing function is preferably bypassed for connection requests originating from all ports at the specified client source **IP address**. In this preferred embodiment, the start affinity notification (as well as an optional end affinity message) is preferably sent from the application to its hosting stack, which forwards the message to the workload balancing function. (Hereinafter, a communication protocol stack on which one or more server applications execute is referred to as a "target stack", a "hosting stack", or a "target/hosting stack". A particular stack may be considered a "target" from the point of view of the workload balancer, and a "host" from the point of view of a server application executing on that stack, or both a target and a host when both the workload balancer and a server application are being discussed.)

Detail Description Paragraph - DETX (18):
[0050] When used as a start affinity request, message format 200 uses fields 202, 204, 206, 208, 210, 212, and 214; fields 216 and 218 are unused. The local **IP address** field 202 preferably specifies the **IP address** for which an affinity is being established. The local port number field 204 specifies the

port number field 204 is zero, then all connection requests arriving at the listening socket (see field 214) are covered by this affinity. If the port number field 204 contains a non-zero value, then the affinity applies only to connection requests arriving for that particular port.

Detail Description Paragraph - DETX (19):
[0051] The partner **IP address** field 206 specifies the source **IP address** of the client to be covered by this affinity. In an optional enhancement, a range of client addresses may be specified for affinity processing. (This enhancement is referred to herein as "affinity group" processing.) In this case, the partner **IP address** field 206 specifies a subnet address, and a subnet mask or prefix field 208 is preferably used to indicate how many **IP addresses** are to be covered. (If the high-order bit is "1", this indicates a subnet mask in normal subnet notation and format. If the high-order is "0", then the value of field 208 indicates how many "1" bits are to be used for the subnet mask.) The partner port number field 210 may specify a particular port number to be used for the affinity, or alternatively may be zero to indicate that the affinity applies to any connection request from the partner **IP address**. (In an alternative embodiment, multiple port numbers may be supported, for example by specifying a comma-separated list of values in field 210.)

Detail Description Paragraph - DETX (21):
[0053] The following verification is preferably performed on the values of the start affinity request message: (1) The local **IP address** value 202 must be a valid **IP address** for which the hosting stack is a valid target for at least one port. (2) The local port value 204, when non-zero, must match an established listening socket. (3) The partner **IP address** 206 must be non-zero. (4) The partner/mask prefix 208 must be non-zero. (5) If the duration 208 exceeds the default maximum for the hosting stack, then the specified value in field 208 will be ignored. (6) If the socket is bound to a specific **IP address,** it must be the same as the local **IP address** in field 202.

Detail Description Paragraph - DETX (22):
[0054] When used as a start affinity response, message format 200 uses all fields shown in FIG. 2A. Most fields are simply copied from the corresponding request message when generating the response message; however, several of the fields are used differently, as will now be described. First, if the local port number 204 was zero on the request message, it will be filled in with an actual port number on the response, as determined by the listening socket handle. Second, the return code field 216 is set, and may indicate a successful start affinity or an unsuccessful start, or perhaps a successful start with a warning message. Finally, the additional information field 218 is set, and preferably conveys additional information about the return code value 216. Preferably, **unique** field value encodings are defined for one or more of the following cases: affinity successfully created; affinity successfully renewed; warning that affinity was not established as requested, and clock was not restarted, because the requested affinity falls within an overlapping affinity for a smaller prefix or larger subnet for which an affinity already exists; unsuccessful because the hosting stack is not a target stack for the specified local **IP address**; unsuccessful because the requested port does not

match the listening socket; unsuccessful because the socket is not a valid listening socket; and unsuccessful because an affinity with the partner **IP address** was already established by another requester.

Detail Description Paragraph - DETX (23):
[0055] Referring now to FIG. 2B, when used as a start affinity request from the hosting stack to the workload balancer, message format 220 uses fields 222, 224, 226, 228, 230, and 232; fields 234 and 236 are unused. Fields 222, 224, 226, 228, and 230 are preferably copied by the hosting stack from the corresponding fields 202, 204, 206, 208, and 210 which were received from the application on its start affinity request message. The local **port** number 224, however, may either have been supplied by the **application** or copied from the listening **socket** information 214. Stack identity 232 identifies the hosting stack to which the subsequent connections covered by the affinity should be sent. The specified value could be a **unique** names within the sysplex (such as an operating system name and a job name within that operating system), or a **unique** address such as an **IP address**; what is required is that the provided identity information suffices to uniquely identify the stack that will handle the incoming connection requests, even if the are multiple stacks per operating system image (such as stacks 34 and 38 in FIG. 1).

Detail Description Paragraph - DETX (24):
[0056] When used as a start affinity response, message format 220 uses all fields shown in FIG. 2B. Preferably, fields 222 through 232 are simply copied from the corresponding request message when generating the response message. The return code field 234 is set in the response, and may indicate a successful start affinity or an unsuccessful start, or a successful start with a warning message. The additional information field 236 is also set, and preferably conveys additional information about the return code value 234. Preferably, **unique** field value encodings are defined for one or more of the following cases: affinity successfully created; affinity successfully renewed; and unsuccessful because an affinity with the partner **IP address** was already established by another requester.

Detail Description Paragraph - DETX (25):
[0057] Preferably, existing affinities that are known to the workload balancing function are stored in a table or other similar structure, such as that illustrated in FIG. 3A. For purposes of illustration but not of limitation, the affinity table may be organized according to the destination server application. As shown in FIG. 3A, the server application type 305 of affinity table 300 preferably comprises (1) the **IP address** 310 of the server application (which corresponds to the destination **IP address** of incoming client connection requests) and (2) the port number 315 of that server application (which corresponds to the destination port number of the incoming client connection requests). These values are taken from fields 222 and 224 of start affinity request messages 220 (FIG. 2B). Preferably, if a server application uses multiple ports, then a separate entry is created in affinity table 300 for each such port. (Alternatively, a list of port numbers may be supported in field 315.)

Detail Description Paragraph - DETX (27):
[0059] Each server application identified by an entry in fields 310, 315 may have an arbitrary number of client affinity entries 325. Each such client affinity entry 325 preferably comprises (1) the client's **IP address** 330 (which corresponds to the source **IP address** of incoming client connection requests), (2) a subnet mask or prefix value 335, which is used for comparing incoming client **IP addresses** to source **IP address** 330 using known techniques, and (3) optionally, the port number 340 of the client application (which corresponds to the source port number of the incoming client connection requests). These values are taken from fields 226, 228, and 230 of start affinity request messages 220 (FIG. 2B). If the client port number is omitted from a particular start affinity message or is set to zero, indicating that an affinity is defined for all ports from a particular client (as discussed above with reference to FIG. 2A), then a port number of zero is preferably used in field 340 to indicate that all ports are to be considered as matching. Alternatively, the port number field 340 may be left blank, or a special keyword such as "ALL" or perhaps a wildcard symbol such as "*" may be provided as the field value. If multiple client port numbers are specified on the start affinity message, then values for the port number field 340 are preferably stored using a comma-separated list (or perhaps an array or a pointer thereto). In an alternative approach, a separate record might be created in the affinity table for each different client port number.

Detail Description Paragraph - DETX (28):
[0060] The table 350 shown in FIG. 3B illustrates a structure that may be used by hosting stacks to manage their existing affinities. As with the table used by the workload balancer and illustrated in FIG. 3 A, entries in the affinity table 350 of FIG. 3B may be organized according to the destination server application. Thus, the server application type 355 of affinity table 350 preferably comprises (1) the **IP address** 360 of the server application and (2) the port number 365 of that server application. These values are taken from fields 202 and 204 of start affinity request messages 200 (FIG. 2A). (Even though the **IP address and port** number of the server **application** are contained in the **socket** control block at the hosting stack, they are preferably stored in the affinity entries as well for efficiency in matching against incoming connection requests.) Preferably, if a server **application** uses multiple **ports,** then a separate entry is created in affinity table 350 for each such **port**.

Detail Description Paragraph - DETX (36):
[0068] When used as an end affinity response from a hosting stack to an application, message format 240 uses all fields shown in FIG. 2C. Fields 242 through 254 are preferably copied from the corresponding request message when generating the response message. The return code field 256 may indicate a successful end affinity or an unsuccessful end. The additional information field 258 is set, and preferably conveys additional information about the return code value 256. Preferably, **unique** field value encodings are defined for one or more of the following cases: affinity successfully ended; unsuccessful, affinity not ended because the requested affinity falls within an overlapping affinity for a smaller prefix or larger subnet for which an

affinity already exists; and unsuccessful because a matching affinity was not found.

Detail Description Paragraph - DETX (37):
[0069] When used as an end affinity indication from a hosting stack to an application, message format 240 uses all fields described for the end affinity response, except that field 256 is not meaningful, and field 258 now contains additional information about the reason for the unsolicited indication message. The additional information field 258 preferably uses **unique** field value encodings for one or more of the following cases to explain why an affinity was ended: timer expiration; the local **IP address** is no longer valid; hosting stack is no longer a target stack for the local **IP address**; and the listening socket was closed.

Detail Description Paragraph - DETX (39):
[0071] When used as an end affinity response from the workload balancer to the hosting stack, message format 260 uses all fields shown in FIG. 2D. Preferably, fields 262 through 272 are simply copied from the corresponding request message when generating the response message. The return code field 274 is set in the response, and may indicate a successful end affinity or an unsuccessful end. The additional information field 276 is also set, and preferably conveys additional information about the return code value 274. Preferably, **unique** field value encodings are defined for one or more of the following cases: affinity successfully ended; unsuccessful end because the specified affinity falls within an affinity for a smaller prefix or larger subnet for which an affinity already exists, and unsuccessful because matching affinity could not be found.

Detail Description Paragraph - DETX (40):
[0072] When used as an end affinity indication from the workload balancer to the hosting stack, message format 260 uses all fields described for the end affinity response, except that field 274 is not meaningful, and field 276 now contains additional information about the reason for the unsolicited indication message. The additional information field 276 preferably uses **unique** field value encodings for one or more of the following cases to inform the hosting stack why the affinity is being ended: the local **IP address** is no longer valid; and the hosting stack is no longer a target stack for the local **IP address**.

Detail Description Paragraph - DETX (52):
[0083] The start affinity message may be sent from the server application to its local hosting stack over a "control socket". As used herein, a control socket is a bi-directional communications socket established between a server application and its hosting stack. Preferably, a server application establishes this control socket when it initializes, along with the normal server listening socket that is used for receiving client requests. However, the control socket provides a control conduit to the server application's hosting TCP/IP stack, rather than a communication vehicle to other applications. Preferably, the destination **IP address and port** number of the server **application** are provided as parameters when establishing the control

socket. Once the control socket is established, the start affinity message (see Block 420), as well as any subsequent end affinity message, is preferably transmitted using that control socket.

Detail Description Paragraph - DETX (54):
[0085] Block 525 then checks to see if it is necessary to notify the workload balancer that this affinity has been started. If the affinity is new (as contrasted to an existing affinity for which a subsequent affinity request has arrived, and which is therefore being renewed by restarting the duration timer), then this test has a positive result and Block 540 adds this target stack's identity information (e.g. its job name and operating system name, or a **unique IP address** associated with the target stack) to a version of the start affinity message that is then forwarded (in Block 550) to the workload balancer. On the other hand, if this affinity is one which is being renewed, and if all timer expiration processing is being handled by the hosting stack, then it is not necessary to forward a (renewing) start affinity message to the workload balancer as no new information would be communicated. In this case, the test in Block 525 has a negative result, and control preferably exits the processing of FIG. 5.

Detail Description Paragraph - DETX (62):
[0093] The logic in FIG. 7 illustrates affinity processing that may be performed when a workload balancer receives incoming client connection requests. A client request is received (Block 705) from a client application (such as client 46 of FIG. 1). The target server application is then determined (Block 710) by examining the destination **IP address** and port number. This information is compared to the workload balancer's stored affinity information (Block 715) to determine if affinities for this application have been defined. With reference to FIG. 3A, this comprises determining whether affinity table 300 has entries 310, 315 matching the destination information from the incoming connection request. If so, then the source **IP address** and port number are compared to the stored affinity information for that application. If an entry for this source **IP address** exists in field 330 of the client affinity information 325 (and matches according to the mask or prefix value stored in field 335), for the target application, and if the source port number of the incoming request either matches a port number specified in field 340 or the entry in field 340 indicates that all port numbers are to be considered as matching, then this is a client request for which a server affinity has been defined. In this case, the test in Block 720 has a positive result, and in Block 730 the target server is selected using the receiving/owning stack field 320; otherwise, when Block 715 fails to find a matching entry in the affinity table, then Block 720 has a negative result and the target server is selected (Block 725) as in the prior art (e.g. using the normal workload balancing process).

Detail Description Paragraph - DETX (79):
[0109] FIG. 11 depicts logic that may be used in the selected target/hosting stack for handling incoming client requests and determining whether port balancing should be performed. At Block 1100, an incoming client request is received. Block 1105 then locates the client **IP address** and port number, and

the destination **IP address** and port number, from that request and checks to see if automatic affinity processing is activated for the target application. If not, then control transfers to Block 1120 which selects an instance as in the prior art. Otherwise, Block 1110 checks the active connections for the target application to determine whether this client already has at least one active connection to that same application. If so, then Block 1125 selects the target application instance to be the same one already in use; otherwise, Block 1120 selects an instance as in the prior art (e.g. using port balancing). In either case, Block 1130 the routes the incoming request to the selected instance, and the processing of FIG. 11 is then complete for this incoming message.

Claims Text - CLTX (4):
3. The method according to claim 2, wherein the selected client is identified by its **Internet Protocol ("IP") address**.

Claims Text - CLTX (5):
4. The method according to claim 2, wherein the selected client is identified by its **Internet Protocol ("IP") address** and port number.

Claims Text - CLTX (12):
11. The system according to claim 10, wherein the selected client is identified by its **Internet Protocol ("IP") address**.

Claims Text - CLTX (13):
12. The system according to claim 10, wherein the selected client is identified by its **Internet Protocol ("IP") address** and port number.

Claims Text - CLTX (20):
19. The computer program product according to claim 18, wherein the selected client is identified by its **Internet Protocol ("IP") address**.

Claims Text - CLTX (21):
20. The computer program product according to claim 18, wherein the selected client is identified by its **Internet Protocol ("IP") address** and port number.